09-语言的接口：语法和程序库，软件设计的发力点

你好！我是郑晔。

在上一讲中，我们学习了程序设计语言模型的演变过程。学习不同的程序设计语言，实质上就是学习不同的编程模型。谈完了模型，接下来，就该说说接口了。

这一讲，我们就来谈谈程序设计语言的接口，一说起程序设计语言的接口，你的直观印象肯定是程序设计语言的语法，那是一个你已经很熟悉的话题了，但程序设计语言还有一个你可能都不曾留意的接口：程序库。

如果你已经能够完成基本功能的开发，想让自己在编程水平上再进一步，成为一个更好的程序员，你就可以从封装程序库开始练习。因为想封装出一个好的程序库所需的能力，就是软件设计所需的能力。封装程序库，可以成为你软件设计进阶的发力点。

消除重复的程序库

我们写程序的时候，只要规模稍微大一点，你就会发现同样的模式经常出现，差别无非是几个地方的参数不一样，这就是重复。

最开始的重复是指令级别的重复，程序员们会把同样的指令序列放到一起，通过传入不同的参数进行识别。你发现了吗？我说的就是函数。函数已经成了今天的主流编程方式，是几乎所有的程序设计语言都有的基础设施，人们甚至忘了它的由来。

写程序的一项主要日常工作就是定义各种函数。一旦你定义了大量的函数，就会发现有很多函数不仅仅在某个项目中是适用的，而且在很多项目中都是适用的。这时，作为一个“懒惰”的程序员，我们就会把这些在多个项目中使用的部分抽取出来，组成一个模块，这就是程序库的来源。

所以，程序库就是为了消除重复而出现的。而消除重复，也是软件设计的初衷。

程序库（Library）是程序员最熟悉的一项内容。学习一门新语言，首先是学习语法（Syntax），然后学习程序库（Library），之后再学习运行时（Runtime），这样，你就具备一门语言的基础了。再往后，你需要了解的就是各种惯用法（Idiom），以及如何运用到实际的工作中。

有一些程序库实在是太常用了，它们就会随着语言一起发布，成为标准库。比如，程序员熟知的第一个程序“Hello, world”的做法来自《C程序设计语言》，其中用到的printf就是来自C的标准库。再比如，Java程序员无人不知的JDK，里面包含了大量的程序库，一个Java程序员如果不能说出几个容器，简直就不好意思和人打招呼。

当然，如果在实际工作中只使用标准库，有些代码写起来还是非常麻烦的。因为标准库提供的能力通常是很基础的。这时，我们就需要利用更多的第三方程序库，它们给提供了更丰富的选项，去完善标准库做得不够的地方。

也就是说，它们会在标准库的基础上，再做一次封装，提供一个新的编程模型，或是新的接口，甚至修正一些标准库的bug，让开发变得更简单。只要是人气足够的语言，在这个方面做得都非常好，它们会提供大量的第三方库。

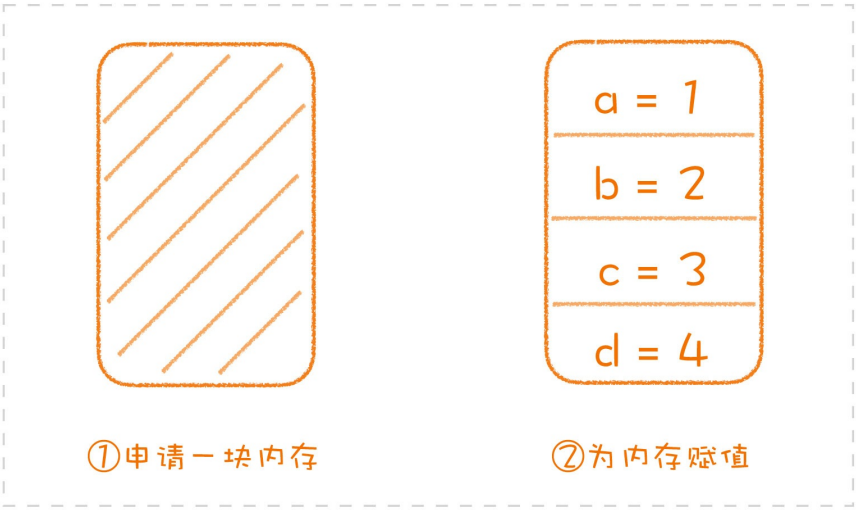
正是因为第三方库的兴起，**怎样管理第三方库**就成了一个问题。今天，这已经成了一个标准的问题，也有了

标准的解决方案，那就是**包管理器**。很多语言都有了自己的包管理器，像Java的Maven、Node的NPM、Ruby的RubyGems等等，而像Rust这样年轻的语言，包管理器甚至变成了随语言的发行包一起发布的一项内容。

语言设计就是程序库设计

new

虽然程序库受限于特定的程序语言，但其表达的思想却不受语言限制。我给你举个例子，在软件开发中，我们最常做的一个操作就是初始化。如果采用C这样比较早期的语言，通常的做法就是，分配一块内存，然后给这块内存赋值，如下图所示：



类似的代码会反复出现，成为一个固定的模式。

而一些新的语言索性就将二者合二为一，成了一个固定的语法，这就是很多人熟悉的new。无论是C++程序员，还是Java程序员，对new一定不陌生。

当我们调用new时会发生什么呢？首先，它会申请一块内存，然后调用对应类的构造函数。类也好，构造函数也罢，这些内容都是C语言没有的，但这个初始化的模式与C语言是如出一辙的。

只不过，在C语言里面，这个操作通常是通过程序库实现的，而到了C++和Java中，它成了语法。一旦变成了语法，它就成了语言的一部分，成为了一个特定的模型。

对于使用者而言，这个模型就是一个接口，只要接口的行为不变，我的代码就不用变。

但对于接口另一面的实现者而言，它就可以做一些特定的工作了。比如，插入不同的内存分配算法，这就是C++的allocator所做的事情；再比如，把内存完全管控起来，这就是Java做的事情。没错，Java之所以能够让程序员忽略内存管理，new功不可没。

一个经过验证的模式最终变成了语言的一部分，而它的起点只是一个常见的用法：一个程序库。

synchronized

我再用Java中的synchronized给你举个例子。我们都知道，并发编程是程序员的一门必修课。学习并发编程，一方面要学习各种概念，比如“锁”；另一方面还要学习不同语言相应的程序库。因为这些概念太常用了。所以，Java干脆把它变成了一个语法，也就是synchronized。

成为语法固然是一个巨大的进步，但是在一些场景下，语法反而会显得僵化。这时候，又轮到程序库登场了。我还以前面所说的new和synchronized为例给你讲解一下。

程序库设计就是语言设计

new

new虽然帮我们解决了一些问题，但与new配合使用的构造函数往往有一个致命伤，那就是它只有一个名字，也就是类的名字。

当我们需要表达多种不同的构造逻辑时，各村就出现了各村的高招。我曾经见过有人利用重载（overload）来解决问题的，不同的构造逻辑用不同类型的参数。比如，一个用HashMap，一个用TreeMap。作为一个新加入项目的程序员，你很难想到这是两种不同的构造逻辑，它们与这里不同类型的数据结构其实一点关系都没有。

一个更好的解决方案是利用**工厂模式**解决这个问题，也就是说，用一个名字更能表意的函数，代替构造函数作为构造入口。

还是以Java为例，ArrayList是Java程序员很熟悉的一个数据结构。如果我要创建一个包含两个元素的ArrayList，同时，还要创建一个初始容量为10的ArrayList。用JDK原生的做法，我可以这么做：

```
// 创建有两个元素的数组
ArrayList<String> listWithElements = new ArrayList();
listWithElements.add("foo");
listWithElements.add("bar");
```

```
// 创建一个初始容量为10的数组。
ArrayList<String> listWithCapacity = new ArrayList(10);
```

这种做法可行，但代码表意并不清晰。那有没有更好的做法呢？Google的Guava程序库，对于同样的场景，给出了一个不同的做法：

```
ArrayList<String> listWithElements = newArrayList("foo", "bar");
ArrayList<String> listWithCapacity = newArrayListWithCapacity(10);
```

显然，从语义上来说，这种做法更清晰。Java领域的行业名著《Effective Java》（第三版）的第一个条款

是“用静态工厂方法代替构造函数”，讨论的就是这种做法。

synchronized

再来看synchronized。它虽然解决了一部分并发的问题，但是，这个解决问题的方式粒度太大了。程序员只要稍微深入一些，就会感到synchronized的掣肘。于是，Java的开发者在Java 5中开始了新一轮的编程模型探索，这次探索的成果就是后来的并发编程库。这也为面试官们提供了新的素材。

你看到了吧，解决同一个问题，它可以用一个语法，也可以用一个程序库，二者之间是等价的。

Andrew Koenig和Barbara Moo写过一本书《C++沉思录》，书里记录了C++早期开发者在设计各种C++特性时的思考，这是一本编程思想之作。当年读这本书时，有两章的标题让我陷入了沉思，分别是**“语言设计就是程序库设计”和“程序库设计就是语言设计”**。

通过今天的学习，这两句话对你来说就不难理解了。因为语法和程序库是在解决同一个问题，二者之间是相互促进的关系。通常是先有程序库，再有语法；如果语法不够好，新的程序库就会出现，新一轮的编程模型就开始孵化。

一切有生命力的语言都会不断改善自己的语法，某些好用的程序库就可以转正成为语法。比如，Java引入Lambda，支持函数式编程；C++引入类型推演，简化了代码编写。

同样地，程序库的发展也在推动着语言的不断进步，有一部分语法就是为了让程序库表现得更好而存在的。比如说：

- C里面的宏，虽然很多人用它来定义常量，但只有编写程序库才能让它更好地发挥出自身的价值；
- Java中的Annotation，很多人都在用，但用它做过设计的人却很少，因为它的使用场景是以程序库居多；
- Scala中的隐式转换，如果你没有设计过DSL，很可能根本就不知道它有什么具体的作用。

至此，你已经能够理解程序设计语言的接口不只包含语法，还有程序库。而且，学习一种程序设计语言提供的模型时，不仅仅要看语法本身有什么，还要了解有语言特性的一些程序库。

所以，对于程序员而言，想要自己的编程水平上一个台阶，学习编写程序库是一个很好的路径。一方面，我们可以锻炼自己从日常工作中寻找重复；另一方面，我们可以更好地理解程序设计语言提供的能力。

总结时刻

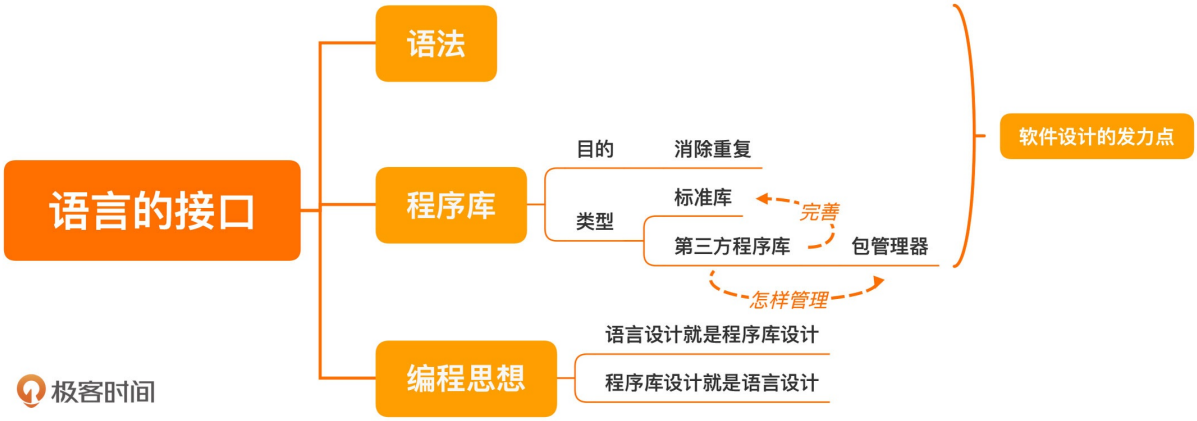
今天，我们的学习主题是程序库，程序库最初只是为了消除重复。后来，逐渐有了标准库，然后有了大量的第三方库，进而发展出包管理器。

如果通用性足够好，一些经过大量实践验证过的程序库甚至会变成语言的语法，而一些语法解决得不够好的地方，又会出现新的程序库去探索新的解决方案。所以，**语言设计就是程序库设计，程序库设计就是语言设计**。二者相互促进，不断发展。

当你开始学习如何编写程序库，你对软件设计的理解就会踏上一个新的台阶。

我们说过，学习不同程序设计语言一个重要的原因是为了相互借鉴。理解了模型和接口，你就知道该借鉴什么，但具体如何借鉴呢？我们还需要了解这些模型是如何实现的，所以下一讲，我们就来谈谈程序设计语言的实现。

如果今天的内容你只能记住一件事，那请记住：**提升软件设计能力，可以从编写程序库入手。**



思考题

今天我们讲到了程序库和语法之间的互相促进，最后，我想请你分享一下，你还能找出哪些语法和程序库互相促进的例子呢？欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这一讲的内容对你有帮助的话，也欢迎把它分享给你的朋友。

精选留言：

- 阳仔 2020-06-12 08:44:40
  程序语言的接口是程序库。我们可以通过学习和开发程序库来提高自己的程序设计能力。
  程序库的开发是为了消除重复。经常遇到的编程模型和重复的功能逻辑都封装成程序库，可以提升编码效率和体验。
  除了使用语言提供的标准程序库还需要理解其它常用的第三方程序库，这些库也是解决某种问题而存在，也是对标准库以及其它三方库的封装。
  很多语言使用包管理器工具管理这些常用库（这也是一个解决问题的模型）
  语言设计和程序设计是相互促进、相互影响的。
  好的编程模型会被整合进语法糖成为语言设计的一部分，而好的语言接口也影响程序设计的模型

- Jxin 2020-06-12 01:21:50
  1.《Effective Java》是本好书，需要尽早一读。
  2.今天记住的是这句话。语言设计就是程序库设计，程序库设计就是语言设计。眼前一亮的感觉。
  3.程序库和语言都属于加法容易，减法难的范畴。除了抽象去重外，其实更重要的是识别什么不该做，以及如何设计向后兼容，易扩展的代码结构。

  4.事实上，很多程序库，为了保证灵活，性能，向后兼容，功能全面。内部代码量会比较大。如果提高设计能力，起步就是程序库，感觉会比较吃力。（写过公共工具包，为此看了hutool和guava等的源码，前者散乱后者优雅，但都不好啃）。在有一定认知后，才能看出一些端倪。
  所以，起步的话，把《重构-既有代码改善》看了，内容简单易懂。然后在项目中持续重构自己的代码，跟自己较真，多思考。每次重构都是在重审自己对设计的理解，循序渐进。